A Distributed Orchestration Algorithm for Edge Computing Resources with Guarantees

A Distributed Orchestration Algorithm for Edge Computing Resources with Guarantees

Gabriele Castellano[†*] Flavio Esposito[†] Fulvio Risso[*]
[†]Computer Science, Saint Louis University, USA
[*]Computer and Control Engineering, Politecnico di Torino, Italy
Email: [†]{gabriele.castellano, flavio.esposito}@slu.edu, [*]{gabriele.castellano, fulvio.risso}@polito.it

***Abstract*—Edge Computing brings flexibility and scalability of virtualization technologies at the edge of the network, enabling service providers to deploy new applications over a richer network infrastructure. However, the coexistence of such variety of applications on the same infrastructure exacerbates the already challenging problem of coordinating resource allocation while preserving the resource assignment optimality. In fact, *(i)* each application can potentially require different optimization criteria due to their heterogeneous requirements, and *(ii)* we may not count on a centralized orchestrator due to the highly dynamic nature of edge networks. To solve this problem, we present DRAGON, a Distributed Resource AssiGnment and OrchestratioN algorithm that seeks optimal partitioning of shared resources between different applications running over a common edge infrastructure. We designed DRAGON to guarantee both a bound on convergence time and an optimal (1-1/e)-approximation with respect to the Pareto optimal resource assignment. We evaluate convergence and performance of DRAGON on a prototype implementation, assessing the benefits compared to traditional orchestration approaches.**

I. Introduction

The emerging Edge Computing paradigm has enabled service providers to supply a large variety of new applications, which benefit from the presence of storage and computing facilities at the edge of the network as well as reduced latency toward end users. Moreover, virtualization technologies enable isolation between different applications that can simultaneously run on separated slices of a shared physical infrastructure.

In cloud-based environments, the deployment of services is often delegated to a centralized component, named *Orchestrator* [1], that usually exploits a *one-size fits-all* policy (e.g., energy saving, number of used nodes, load balancing) to decide (*i*) where to place service components, (*ii*) how many resources have to be assigned to each of them [2] and (*iii*) the set of metrics/events signaling that the service has to be rescheduled (e.g., because of an unexpected load increase).

However, such a centralized approach may be sub-optimal or not applicable at the edge of the network. First, such environment may be characterized by high churn rates, unpredictable changes in the network topology and even temporary network partitions; this may favor distributed orchestration approaches against a centralized orchestrator, which may not be even reachable. Second, the largely heterogeneous set of applications running at the edge of the network may have diverse and unpredictable objectives, not to mention the necessity to react differently to the same event, such as a load increase. In the above circumstance, some applications may scale up/out, other may modify the service behavior (e.g., choose a more aggressive video transcoder), other may migrate, and more; the above behaviors are difficult to achieve with a one-size fits-all orchestrator.

For instance, the optimization of a Content Distribution Network (CDN) may require to monitor the average miss-rate that users experience on deployed caches, which can be used to recognize occasional hot spots (e.g., flash crowd during live events); this in turn requires optimizing the service by relocating (and possibly duplicating) some caches. Vice versa, an online gaming application featuring assisted migration for user mobility may want to relocate part of the application [3] to reduce latency, possibly identifying a convenient time frame (e.g., after a checkpoint) for this operation.

Distributed content delivery and caching, Internet of Things, disaster response, vehicle-to-everything automotive and video acceleration are only some of the multitude of applications [4] that can benefit from being deployed at the edge of the network. However, coordinating such a plethora of applications, each one featuring different policies and deployment approaches, without relying on a centralized orchestrator, brings to light several challenges. How could several processes, each operating with different goals and policies, converge to a globally optimal resource management over a shared edge infrastructure? How could we avoid violations of global policies or feasibility constraints of several coexisting applications? How can we guarantee convergence to a distributed resource allocation agreement and performance optimality given the NP-hard [5] nature of the service placement problem?

To answer these questions, we present DRAGON, an asynchronous Distributed Resource AssiGnment and OrchestratioN algorithm. DRAGON leverages the max-consensus literature and the theory of submodular functions to enable a set of applications, featuring diverse objectives and optimization metrics, to reach an agreement on how infrastructure resources have to be (temporary) assigned, without the necessity of a centralized orchestrator. Our contributions are as follows:

Design contributions. We introduce the *Applications-Resources Assignment Problem* and use linear programming to model its objective and constraints (Sections III). Finding a centralized optimal solution is often infeasible even for a single optimizer. We use the solution to the centralized problem as a baseline global optimal to show DRAGON's performance optimality guarantees.

Algorithmic contributions. We detail our DRAGON asynchronous algorithm (Sections IV and V) and we show how it provides guarantees on both convergence time and expected resource assignment performance to a set of independent edge applications (Section VI).

Evaluation contributions. We evaluate both performance scalability and convergence properties of DRAGON, comparing them with the traditional one-size fits-all approaches. Moreover, we assess DRAGON's benefits analyzing as reference use cases *(i)* the problem of cache placement for a CDN provider and *(ii)* the edge migration for mobile gaming (Section VII). Our findings confirm the applicability of this approach in edge infrastructures and the performance advantages over traditional one-size fits-all orchestration approaches.

II. Related Work

Optimization of edge applications. Recent works [6]–[8] propose ad-hoc optimization, each one targeting a single edge application separately. For instance, [6] optimizes the placement of roadside units on new generation vehicular networks. Instead, [7] focuses on the service placement problem in mobile applications, where the dynamism of user's location plays a key role. Finally, [8] proposes an optimal allocation for high-performance video streaming in 5G networks. While above solutions enable optimization for isolated applications, at the best of the author's knowledge there have been no studies about how such a variety of service embedding algorithms can coexist on a shared infrastructure without undermining the overall performance optimality.

Distributed resource assignment. Another related set of solutions concerns the partitioning of shared resources. Catena [9] proposes an approach oriented to multi-objective infrastructure providers, without analyzing the orthogonal problem of optimizing application utilities. In cloud environments, Mesos [10] enables dynamic resource partitioning and allows the coexistence of diverse cluster computing frameworks, each one featuring different scheduling needs. It exploits a master that assigns resources dynamically by making offers to demanding frameworks. However, mandating the existence of such a component may not be suitable in a scenario where services are executed on scattered compute nodes, e.g., at the edge of the network, which features arbitrary and dynamic topologies. In this context, we should rely on solutions that provide decentralized consensus (e.g., Paxos [11] and Raft [12]) to reach agreement on resource assignment. However, none of them simultaneously provides *(i)* guarantees on convergence time and performance, and *(ii)* a fully distributed approach.

III. Problem Definition and Modeling

This section defines the (NP-hard) *applications-resources assignment problem* by leveraging linear programming.

Let us model an *application* as a multiset — a set in which element repetition is allowed — whose elements are selected among N_s (abstract) services to be embedded on a shared (physical) edge infrastructure. A *service* is an abstract instance of a physical function, e.g., a load balancer, a video transcoder or a content cache, which can be implemented by selecting the best possible physical *function* among the N_f available ones. In fact, functions may feature different characteristics such as execution environment (virtual machine, container, dedicated hardware), required resources, or provided level of QoS.

The infrastructure is partitioned in N_v hosting nodes, each one with potentially different physical capacities. We assume that each function consumes a given amount of resources such as CPU, storage, memory, network bandwidth, etc., which are modeled with N_ρ different types.

Finally, let us consider N_a applications, all simultaneously demanding resources from a shared edge infrastructure, each one following a potentially different optimization strategy. We assume that the application itself will select the best (feasible) functions that are required to implement its composing services, then allocate them in the most appropriate location.

Our goal is to maximize a global utility U while finding an infrastructure-bounded applications-resources assignment that allows the deployment of each application. We define an applications-resources assignment to be *infrastructure-bounded* if the consumption of all assigned functions allocated on each hosting node does not exceed the $\boldsymbol{\rho}_n$ available resources on that node.

We model the applications-resources assignment problem with an integer program; its binary decision variable x_{ijn} is equal to one if an instance of function j has been assigned to application i on hosting node n and to zero otherwise.

$$\text{maximize} \quad \sum_{i=1}^{N_a}\sum_{j=1}^{N_f}\sum_{n=1}^{N_v} U_{ijn}(\boldsymbol{x}_i)x_{ijn} \tag{1.1}$$

subject to

$$\sum_{i=1}^{N_a}\sum_{j=1}^{N_f} x_{ijn}c_{jk} \leq \rho_{nk} \qquad \forall k \in \mathcal{K},\ \forall n \in \mathcal{N} \tag{1.2}$$

$$\sum_{j=1}^{N_f}\sum_{n=1}^{N_v} x_{ijn} = \sum_{m=1}^{N_s}(\sigma_{im})y_i \qquad \forall i \in \mathcal{I} \tag{1.3}$$

$$\sum_{j=1}^{N_f}\left(\sum_{n=1}^{N_v} x_{ijn}\right)\lambda_{mj} \geq y_i \qquad \forall m \in \mathcal{M},\ \forall i \in \mathcal{I} \tag{1.4}$$

$$\sum_{n=1}^{N_v} x_{ijn} \leq 1 \qquad \forall j \in \mathcal{J},\ \forall i \in \mathcal{I} \tag{1.5}$$

$$\sum_{j=1}^{N_f} x_{ij} \geq 1 - N_f y_i \qquad \forall i \in \mathcal{I} \tag{1.6a}$$

$$\sum_{j=1}^{N_f} x_{ij} \leq 1N_f y_i \qquad \forall i \in \mathcal{I} \tag{1.6b}$$

$$x_{ijn} \in \{0,1\} \qquad \forall (i,j,n) \in \mathcal{I}\times\mathcal{J}\times\mathcal{N} \tag{1.7a}$$

$$y_i \in \{0,1\} \qquad \forall i \in \mathcal{I} \tag{1.7b}$$

$$c_{jk} \in \mathbb{N} \qquad \forall (j,k) \in \mathcal{J}\times\mathcal{K} \tag{1.7c}$$

$$\rho_{nk} \in \mathbb{N} \qquad \forall (n,k) \in \mathcal{N}\times\mathcal{K} \tag{1.7d}$$

$$\lambda_{mj} \in \{0,1\} \qquad \forall (m,j) \in \mathcal{M}\times\mathcal{J} \tag{1.7e}$$

$$\sigma_{im} \in \{0,1\} \qquad \forall (i,m) \in \mathcal{I}\times\mathcal{M} \tag{1.7f}$$

where $\boldsymbol{x}_i \in \{0,1\}^{N_f \times N_v}$ is the *assignment vector* for application i, whose $j^{th} \times n^{th}$ element is x_{ijn}. The auxiliary variables y_i are equal to 1 if at least an instance of any

function has been assigned to application i, and 0 otherwise (constraints 1.6a, 1.6b, 1.7b). The index sets are defined as $\mathcal{I} \triangleq \{1,\dots,N_a\}$, $\mathcal{M} \triangleq \{1,\dots,N_s\}$, $\mathcal{J} \triangleq \{1,\dots,N_f\}$, $\mathcal{K} \triangleq \{1,\dots,N_\rho\}$ and $\mathcal{N} \triangleq \{1,\dots,N_v\}$. Variable ρ_{nk} represents the amount of resource k available on node n; furthermore, we denote $\boldsymbol{\rho}_n \in \mathbb{N}_n^N$ the overall capacity of node $n \in \mathcal{N}$. With $c_{jk} \in \mathbb{N}$ we capture the cost of function j in terms of resource k; thus, we name $\boldsymbol{c}_j \in \mathbb{N}_\rho^N$ the cost vector of function $j \in \mathcal{J}$. We set $\lambda_{mj} = 1$ if the abstract service m can be implemented (i.e., deployed) through function j, while $\sigma_{im} = 1$ if application i needs service component m.

The *utility function* models the overall gain $U_{ijn}(\boldsymbol{x}_i)$, i.e., the utility that the system gains by assigning $\boldsymbol{c}_j$ resources to application i, allowing it to add function j to its assignment vector $\boldsymbol{x}_i$. Note that the gain does not depend merely from the service itself; in fact, it depends *(i)* on which function is used to instantiate a specific service and *(ii)* on which node the chosen function is deployed. Note how constraint (1.2) ensures that the solution is infrastructure-bounded, while constraints (1.3 and 1.4) avoid partial allocations.

IV. Single-Node DRAGON

In this section we introduce DRAGON (Distributed Resource AssiGnment and OrchestratioN), a novel approximation algorithm that we designed to solve the NP-hard Problem 1 through a distributed approach.

Each application i runs a DRAGON agent, which starts a voting procedure with the aim of acquiring the resources needed to deploy its assignment vector $\boldsymbol{x}_i$, and participates to a resource election protocol. Voting and elections are performed at the node level. Applications that are "elected", i.e., that they win the distributed assignment problem, gain the right to allocate their demanded amount of (virtual) resources on a certain (physical) node. In the first phase, each agent performs the election locally, based on its local state awareness. Then a max-consensus based distributed agreement phase guarantees the converge of the election process.

To describe all core mechanisms of our approach, we first introduce a simplified Single-Node version (SN-DRAGON), featuring a single hosting node on the underlying infrastructure. Note that, in SN-DRAGON, structures introduced in Section III are simplified by the absence of the node index n.

To describe the algorithm, we give the following definitions:

Definition 1. *(private utility function $\boldsymbol{u}_i$). Given a set $\mathcal{I}$ of applications and a set $\mathcal{J}$ of functions, we define private utility function of application $i \in \mathcal{I}$, and we denote it with $\boldsymbol{u}_i \colon \mathcal{J} \to \mathbb{R}$, the utility $u_{ij} \in \mathbb{R}$ that application i gains by adding function $j \in \mathcal{J}$ to its assignment vector $\boldsymbol{x}_i$, i.e., implementing one of its services through the function j.*

Each application may have a different (conflicting) objective and may have no incentive to disclose its utility; however, our model, and so our algorithm, maximizes a global objective (Equation 1.1), that in DRAGON is a policy. Since we assume that a Pareto optimality is sought, the global utility is a function of the applications private utilities, i.e.,

$$\boldsymbol{U}_i(\boldsymbol{x}_i) = f(\boldsymbol{u}_i(\boldsymbol{x}_i)),\ \forall i \in \mathcal{I}.$$

DRAGON needs a vote vector that we define as follows.

Definition 2. *(vote vector $\boldsymbol{v}^i$). Given a distributed voting process among a set $\mathcal{I}$ of N_a applications, we define $\boldsymbol{v}^i \in \mathbb{R}_+^{N_a}$ to be the vector of current winning votes known by application $i \in \mathcal{I}$. Each element v_ι^i is a positive real number representing the vote of $\iota \in \mathcal{I}$ known by application i, if i thinks that ι is a winner of the election phase. Otherwise, v_ι^i is 0.*

Since applications compute resource assignment in a distributed fashion, they could possibly have different views until an agreement on the election winner(s) is reached; we use the apex i to refer to the vote vector as seen by application i at each point in the agreement process. During the algorithm description, for clarity, we omit the apex i when we refer to the local vector (the same applies also for the following vectors).

Definition 3. *(demanded resource vector $\boldsymbol{r}^i$). Given a voting process among a set $\mathcal{I}$ of N_a applications on N_ρ different types of shared resources, we define as demanded resource vector $\boldsymbol{r}^i \in \mathbb{N}_+^{N_a \times N_\rho}$, the vector of total resources currently demanded by each application; each element $\boldsymbol{r}_\iota^i \in \mathbb{N}^{N_\rho}$ is the amount of resources demanded by application $\iota \in \mathcal{I}$ with its most recent vote v_ι^i known by $i \in \mathcal{I}$.*

Definition 4. *(voting time vector $\boldsymbol{t}^i$). Given a set $\mathcal{I}$ of N_a applications participating to a distributed voting process, we define as voting time vector $\boldsymbol{t}^i \in \mathbb{R}_+^{N_a}$, the vector whose element t_ι^i represents the time stamp of the last vote v_ι^i known by $i \in \mathcal{I}$ for application $\iota \in \mathcal{I}$.*

We also give the following definition of neighborhood:

Definition 5. *(neighborhood $\bar{\mathcal{I}}_i$). Given a set $\mathcal{I}$ of applications, we define neighborhood $\bar{\mathcal{I}}_i \subseteq \mathcal{I} \setminus \{i\}$ of application $i \in \mathcal{I}$, the subset of applications directly connected to i.*

The notion of neighborhood is generalizable with the set of agents reachable within a given latency upper bound. We are now ready to describe SN-DRAGON (Algorithm 1).

Algorithm Overview. On each application i, the DRAGON agent runs an *Orchestration Phase* (Algorithm 2) where an optimal assignment, if any, is built and voted to participate in the resource election. Votes here are updated in a distributed election process. If any value of the vote vector $\boldsymbol{v}^i$ is changed, i sends its vectors $\boldsymbol{v}^i$, $\boldsymbol{r}^i$ and $\boldsymbol{t}^i$ to its (first-hop) neighbors, then waits for a response coming from any number of them. During the *Agreement Phase*, all vectors $\boldsymbol{v}^{i'}$, $\boldsymbol{r}^{i'}$ and $\boldsymbol{t}^{i'}$ received from neighbor i' are used in combination with the local values (Algorithm 5), to reach an agreement with i'.

Note that the assignment vector $\boldsymbol{x}_i$ of each application i does not need to be exchanged. Agents are aware of the resources demand from their peers, but are unaware of the details regarding which functions they wish to allocate.

The remainder of this section gives more details on the two main phases of the SN-DRAGON algorithm.

Algorithm 1 `SN-DRAGON` for application i at iteration t

1: `orchestration`$(\boldsymbol{v}(t-1),\ \boldsymbol{r}(t-1),\ \boldsymbol{\rho})$
2: **if** $\exists \iota \in \mathcal{I}:\ v_\iota(t) \neq v_\iota(t-1)$ **then**
3: `send`$(i',\ t),\ \forall i' \in \bar{\mathcal{I}}_i$
4: `receive`$(i',\ t),\ \forall i' \in \bar{\mathcal{I}}_i$
5: `agreement`$(i',\ t),\ \forall i' \in \bar{\mathcal{I}}_i$

Algorithm 2 `orchestration` for application i at iteration t

Input: $\boldsymbol{v}(t-1),\ \boldsymbol{r}(t-1),\ \boldsymbol{t}(t-1),\ \boldsymbol{\rho},\ \boldsymbol{c}$
Output: $\boldsymbol{v}(t),\ \boldsymbol{r}(t),\ \boldsymbol{t}(t)$
1: **if** $t \neq 0$ **then**
2: $\boldsymbol{v}(t),\ \boldsymbol{r}(t),\ \boldsymbol{t}(t) = \boldsymbol{v}(t-1),\ \boldsymbol{r}(t-1),\ \boldsymbol{t}(t-1)$
3: **do**
4: $\bar{v}_i = v_i(t)$
5: **if** $v_i(t-1) \neq 0\ \wedge\ v_i(t) = 0$ **then** ▷ outvoted
6: `embedding`(t) ▷ find next $\boldsymbol{x}_i$ maximizing $\boldsymbol{u}_i$
7: `voting`$(\boldsymbol{x}_i,\ \boldsymbol{c})$ ▷ vote $\boldsymbol{x}_i$ using $\boldsymbol{U}$
8: `election`$(\boldsymbol{v}(t),\ \boldsymbol{r}(t),\ \boldsymbol{\rho})$
9: **while** $\bar{v}_i \neq v_i(t)$ ▷ repeat until not outvoted

A. Orchestration Phase

After the initialization of local vectors $\boldsymbol{v}(t)$, $\boldsymbol{r}(t)$ and $\boldsymbol{t}(t)$ for the current iteration t (Algorithm 2, line 2), each DRAGON agent uses Algorithm 2, line 8 to elect the current winners according to the known votes updated at the last iteration. If agent i has been outvoted (Algorithm 2, line 5), the algorithm starts to iterate among *(i)* an *embedding routine* (Algorithm 2, line 6), which computes the next suitable assignment vector $\boldsymbol{x}_i$ maximizing i's private utility, *(ii)* a *voting routine* (Algorithm 2, line 7) where agent i votes for the resources that follow the last computed assignment vector and *(iii)* the *election routine* (Algorithm 2, line 8).

The iteration continues until agent i does not get outvoted anymore (Algorithm 2, line 9). This may happen if either *(i)* the selected assignment vector allows i to win the election or *(ii)* there are no more suitable assignments $\boldsymbol{x}_i$ (then no new vote has been generated).

Remark. *To guarantee convergence, DRAGON forbids outvoted applications to re-vote with an higher utility value on resources that they have lost in past rounds. Re-voting is, however, allowed only on residual resources.*

Note that an asynchronous agreement may never terminate unless we forcefully timeout the consensus process. However, we use the theory of max-consensus to show that the agreement phase stops as long as we have reliable communication.

1) Embedding Routine: Either during the first iteration ($t = 0$), or any time application i is outvoted, SN-DRAGON invokes an embedding routine (Algorithm 2, line 6) that, based on the private policies of i, computes the next best suitable assignment vector $\boldsymbol{x}_i$. Therefore, this routine is in turn private for each application, and strictly dependent from the specific nature of the application itself (each of them may follow a different deployment strategy, seek optimization of specific metrics and even feature additional deployment constraints).

2) Voting Routine: After a new assignment vector has been built, each DRAGON agent runs a voting routine, updating the time of its most recent vote; if the assignment vector is valid,

Algorithm 3 `voting` for application i at iteration t

Input: $\boldsymbol{x}_i,\ \boldsymbol{c}$
Output: $v_i(t), \boldsymbol{r}_i(t), t_i(t)$
1: $t_i(t) = t$ ▷ vote time
2: **if** $\boldsymbol{x}_i \neq \mathbf{0}$ **then** ▷ valid assignment
3: $r_{ik}(t) = \Sigma_j x_{ij} c_{jk},\ \forall k \in \mathcal{K}$ ▷ demanded resources
4: $v_i(t) =$ `score`$(\boldsymbol{x}_i)$ ▷ vote new assignment

Algorithm 4 `election` routine at iteration t

Input: $\boldsymbol{v}(t),\ \boldsymbol{r}(t),\ \boldsymbol{\rho}$
Output: $\boldsymbol{v}(t)$
1: $\bar{\boldsymbol{\rho}} = \boldsymbol{\rho}$ ▷ residual resources
2: $\mathcal{W} = \varnothing$ ▷ winner set
3: **do**
4: $\mathcal{I}_b = \{i \in \mathcal{I} |\ r_{ik}(t) \leq \bar{\rho}_k, \forall k \in \mathcal{K}\}$ ▷ valid candidates
5: $\omega = \arg\max_{i \in \mathcal{I}_b \setminus \mathcal{W}} \left\{ \frac{v_i(t)}{\|\boldsymbol{r}_i(t)\|} \right\}$ ▷ candidate with higher vote
6: $\mathcal{W} = \mathcal{W} \cup \{\omega\}$ ▷ add to winners
7: $\bar{\rho}_k = \bar{\rho}_k - r_{\omega k},\ \forall k \in \mathcal{K}$ ▷ decrease residual resources
8: **while** $\mathcal{I}_b \setminus \mathcal{W} \neq \varnothing$ ▷ repeat until no candidate remains
9: $v_\iota = 0,\ \forall \iota \in \mathcal{I} \setminus \mathcal{W}$ ▷ reset loser votes

all demanded resources are updated and voted, through a *score function* derived by the global utility (Algorithm 3). Although the raw global utility itself may be used as score function to compute votes, in Section V-A we give recommendation on how to properly wrap it to guarantee convergence and optimal approximation bound (Section VI).

3) Election Routine: The last step of the *Orchestration Phase* (Algorithm 2, line 8) is a resource election that decides which applications are capable of allocating the demanded resources on the shared hosting node (Algorithm 4). Based on the most recent known votes $\boldsymbol{v}(t)$, the related resource demands $\boldsymbol{r}(t)$ and the capacity $\boldsymbol{\rho}$ of the shared node, this procedure selects applications by mean of a greedy approach. At each step, it *(i)* discards any application whose demanded resources $\boldsymbol{r}_i$ exceed the residual node capacity and *(ii)* selects the one with the highest ratio vote to demanded resources (Algorithm 4, lines 4-5). The elected one is then added to the winners set and the amount of resources assigned to the new winner are removed from the residual set (Algorithm 4, lines 6-7). The greedy election ends when either all candidates result winners, or residual resources are not enough for any of those remaining. Finally, votes of applications that did not win the election are reset (Algorithm 4, line 9). In Section VI we show that the greedy heuristic gives guarantees on the optimal approximation.

B. Agreement Phase

Once received vectors $\boldsymbol{v}^{i'}$, $\boldsymbol{r}^{i'}$ and $\boldsymbol{t}^{i'}$ from every i' in its neighborhood, each agent runs an Agreement Phase. During this phase, applications make use of a consensus mechanism to reach an agreement on their vote vector $\boldsymbol{v}^i$, hence on the overall resources assignment (Algorithm 5). By adapting the definition of *consensus* [13] to the applications-resources assignment problem, we define our own notion of consensus (election) as follows:

Definition 6. *(consensus (election)). Let us consider a set $\mathcal{I}$ of N_a applications sharing a computing edge infrastructure*

Algorithm 5 `agreement` with application i' at iteration t

Input: $\boldsymbol{v}(t),\ \boldsymbol{r}(t),\ \boldsymbol{t}(t),\ \boldsymbol{v}^{i'}(t),\ \boldsymbol{r}^{i'}(t),\ \boldsymbol{t}^{i'}(t)$
Output: $\boldsymbol{v}(t),\ \boldsymbol{r}(t),\ \boldsymbol{t}(t)$

for all $\iota \in \mathcal{I}$ **do**
  if $t_\iota(t) < t_\iota^{i'}(t)$ **then** ▷ received newer vote
    $v_\iota(t) = v_\iota^{i'}(t)$
    $r_{\iota k}(t) = r_{\iota k}^{i'}(t),\ \forall k \in \mathcal{K}$
    $t_\iota(t) = t_\iota^{i'}(t)$

through an election routine driven by, for each application $i \in \mathcal{I}$, the vote vector $\boldsymbol{v}^i(t) \in \mathbb{R}_+^{N_a}$, the demanded resource vector $\boldsymbol{r}^i(t) \in \mathbb{R}_+^{N_a \times N_\rho}$ and the voting time vector $\boldsymbol{t}^i(t) \in \mathbb{N}^{N_a}$. Let $e : \mathbb{R}_+^{N_a}, \mathbb{N}^{N_a \times N_\rho} \to 2^{\mathcal{I}}$ be the election function, that given a vote vector $\boldsymbol{v}$ and the demanded resources $\boldsymbol{r}$ gives a set of winners. Given the consensus algorithm for application i at iteration $t+1$, $\forall \iota \in \mathcal{I}$,

$$v_\iota^i(t+1) = v_\iota^{i'}(t),\ \boldsymbol{r}_\iota^i(t+1) = \boldsymbol{r}_\iota^{i'}(t), \quad \text{with } i' = \underset{i' \in \bar{\mathcal{I}}_i \cup \{i\}}{\arg\max}\{t_\iota^{i'}(t)\}, \tag{2}$$

consensus (election) *among the applications is said to be achieved if $\exists \bar{t} \in \mathbb{N}$ such that, $\forall t \geq \bar{t}$ and $\forall i, i' \in \mathcal{I}$,*

$$\begin{cases} e(\boldsymbol{v}^i(t), \boldsymbol{r}^i(t)) \equiv e(\boldsymbol{v}^{i'}(t), \boldsymbol{r}^{i'}(t)) \\ \boldsymbol{v}_\iota^i(t) \neq 0 \iff \iota \in e(\boldsymbol{v}^i(t)),\ \forall \iota \in \mathcal{I}, \end{cases} \tag{3}$$

i.e., on all applications the election function computes the same winners set and only winner votes are non zero.

Being DRAGON asynchronous by design, at each iteration t the agreement phase can start even if agents have received a vote message from only a subset of their neighbors.

V. Multi-Node DRAGON Algorithm

In this section we extend the SN-DRAGON approximation algorithm by proposing a distributed multi-node solution to Problem 1, that we merely call DRAGON. All data structures given in the previous section are extended with a new index $n \in \mathcal{N}$, where $\mathcal{N}$ is the set of compute nodes and $|\mathcal{N}| = N_v$. For example, the *vote vector* in Definition 2 is extended with $\boldsymbol{v}^i \in \mathbb{R}_+^{N_o \times N_v}$, where each element $v_{\iota n}^i$ is the last vote of application $\iota \in \mathcal{I}$ on node $n \in \mathcal{N}$ as known by i.

As in SN-DRAGON, DRAGON iterates between an *Orchestration Phase* and an *Agreement Phase*. While the *Agreement Phase* is identical, despite being repeated for each node $n \in \mathcal{N}$, some procedures of the *Orchestration Phase* are extended.

During the *Orchestration Phase* of DRAGON, an *embedding routine* selects, for each service needed by the application, both the function $j \in \mathcal{J}$ that should be used to implement it and the node $n \in \mathcal{N}$ where j should be placed. The *voting routine* is repeated once for each node involved in the current assignment $\boldsymbol{x}_i$, so that a vote $v_{in}(t)$ is generated for every n.

Remark. *In DRAGON an assignment $\boldsymbol{x}_i$ is considered valid only if application i wins all elections on each node n involved in the assignment $\boldsymbol{x}_i$. If any election is lost, DRAGON resets the vote vector and a new assignment is built from scratch to avoid suboptimal assignments.*

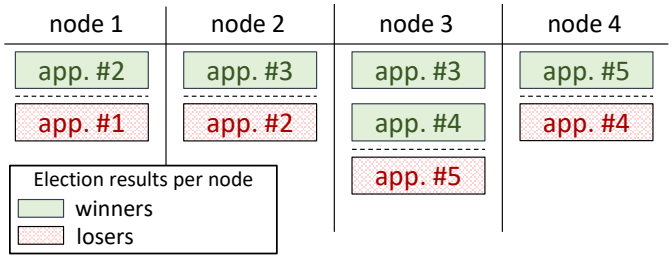


Fig. 1: Example of false winners after an election routine: application #2 prevents #1 to allocate needed resources on node 1, although #2 cannot be deployed, since it lost elections on node 2.

In DRAGON, the election routine features a conflict resolution named *election-recount*, which handles potential suboptimality deriving as a result of elections. Consider the assignment scenario in Figure 1; most resources of node 1 have been assigned to app. #2, thus preventing the deployment of app. #1; however, having #2 lost the election on node 2, it will release its previous vote on node 1 at the next iteration. Therefore, app. #1 could be considered a winner.

The election-recount subroutine copes with this problem by identifying which applications should be removed from the election so that the solution results optimized (the description of the subroutine is omitted due to lack of space).

A. Recommendations on the score function

DRAGON's score function is a policy. Many policies may work well in practice, but in some cases they may lead to arbitrarily bad performance. As we will see in the next section, DRAGON guarantees both convergence and a given performance lower bound as long as the function maximized during the election routine is submodular. In this section we give recommendation on the score function $\mathcal{V}$ that each application should use during the voting routine described in Algorithm 3 to satisfy this property. Analytic results are shown in the next section.

Let $\mathcal{U}_{in}(\boldsymbol{x}_i) = \Sigma_j U_{ijn}(\boldsymbol{x}_i) x_{ijn}$ be the overall node utility of application i on node n. To guarantee convergence of the election process, we let each peer i communicate its vote on node n obtained from the score function:

$$\mathcal{V}_i(\boldsymbol{x}_i, \mathcal{W}_n, n) = \min_{\omega \in \mathcal{W}_n} \{\mathcal{U}_{in}(\boldsymbol{x}_i), \mathcal{S}_{in}(\omega)\}, \tag{4}$$

where $\mathcal{W}_n \subseteq \mathcal{I}$ is the current winner set for node n, i.e., $v_{\omega n}(t) \neq 0\ \forall \omega \in \mathcal{W}_n$, and $\mathcal{S}_{in}$ is defined as

$$\mathcal{S}_{in}(\omega) = \begin{cases} +\infty & \text{if } i \text{ never voted on } n, \\ \|\boldsymbol{r}_{in}(t)\| \frac{v_{\omega n}(t)}{\|\boldsymbol{r}_{\omega n}(t)\|} & \text{otherwise.} \end{cases}$$

Since $\mathcal{U}_{in}(\boldsymbol{x}_i) \geq 0$ by definition, if i computes each vote with the function $\mathcal{V}$, it follows that, $\forall (i,n) \in \mathcal{I} \times \mathcal{N}$, $\mathcal{V}_i(\boldsymbol{x}_i, n) \geq 0$. Note how, if it is not the first time that i votes on n, the vote $v_{in}(t)$ generated at iteration t never results as an outvote of any application that has been previously elected on node n, during the election process described in Algorithm 4.

VI. Convergence and Performance Guarantees

In this section we present results on the convergence properties of our DRAGON distributed approximation algorithm. As

in Definition 6, by convergence we mean that a valid solution to the applications-resources assignment problem is found in a finite number of steps. Moreover, starting from well-known results on submodular functions, in this section we show that DRAGON guarantees an $(1-e^{-1})$-approximation bound, and that this bound is also optimal, i.e. there is no better guarantee, unless $NP \subseteq DTIME(n^{O(\log\log n)})$.

Note that, if (4) is used as score function, the election routine of DRAGON is equivalent to a greedy algorithm attempting to find, for each node n, the set of winner applications $\mathcal{W}_n \subseteq \mathcal{I}$ such that the set function $z_n : 2^{\mathcal{I}} \to \mathbb{R}$, defined as

$$z_n(\mathcal{W}_n) = \sum_{\omega \in \mathcal{W}_n} \mathcal{V}_\omega(\boldsymbol{x}_\omega, \mathcal{W}_n, n), \tag{5}$$

is maximized. By construction of $\mathcal{V}$, we have that z_n is monotonically non-decreasing and $z(\varnothing) = 0$.

Definition 7. *(submodular function). A set function $z : 2^{\mathcal{I}} \to \mathbb{R}$ is submodular if and only if, $\forall \iota \notin \mathcal{W}' \subset \mathcal{W}'' \subseteq \mathcal{I}$,*

$$z(\mathcal{W}'' \cup \{\iota\}) - z(\mathcal{W}'') \leq z(\mathcal{W}' \cup \{\iota\}) - z(\mathcal{W}'). \tag{6}$$

This means that the marginal utility of adding ι to the input set, cannot increase due to the presence of additional elements. Next we show that the total score z_n (5) is submodular. Our intuition behind its submodularity is that the score function $\mathcal{V}_n$ can, at most, decrease due to the presence of additional elements in $\mathcal{W}_n$. Formally, we have:

Lemma VI.1. *z_n (5) is submodular.*

Proof: Since $\mathcal{W}'_n \subset \mathcal{W}''_n$, we have

$$\min_{\omega \in \mathcal{W}''_n} \left\{ \|\boldsymbol{r}_{\iota n}(t)\| \frac{v_{\omega n}(t)}{\|\boldsymbol{r}_{\omega n}(t)\|} \right\} \leq \min_{\omega \in \mathcal{W}'_n} \left\{ \|\boldsymbol{r}_{\iota n}(t)\| \frac{v_{\omega n}(t)}{\|\boldsymbol{r}_{\omega n}(t)\|} \right\},$$

and so, for (4),

$$\mathcal{V}_\iota(\boldsymbol{x}_i, \mathcal{W}''_n, n) \leq \mathcal{V}_\iota(\boldsymbol{x}_i, \mathcal{W}'_n, n). \tag{7}$$

By definition of z_n, the marginal gain of adding ι to $\mathcal{W}_n$ is

$$z_n(\mathcal{W}_n \cup \{\iota\}) - z_n(\mathcal{W}_n) = \mathcal{V}_\iota(\boldsymbol{x}_i, \mathcal{W}_n, n), \forall \iota \notin \mathcal{W}_n \subseteq \mathcal{I},$$

therefore, substituting in (7), we have the claim. ∎

Convergence Guarantees. A necessary condition for convergence in DRAGON is that all applications are aware of which are the winning votes for an hosting node. This information needs to traverse all applications in the communication network (at least) once.

The communication network of a set of applications $\mathcal{I}$ is modeled as an undirected graph, with unitary length edges between each couple $i', i'' \in \mathcal{I}$ such that $i'' \in \bar{\mathcal{I}}_{i'}$ and $i' \in \bar{\mathcal{I}}_{i''}$, being $\bar{\mathcal{I}}_{i'} \subseteq \mathcal{I} \setminus \{i'\}$ and $\bar{\mathcal{I}}_{i''} \subseteq \mathcal{I} \setminus \{i''\}$ respectively the neighborhoods of i' and i''.

Theorem VI.2. *(Convergence of synchronous DRAGON). Consider an infrastructure of N_v hosting nodes, whose resources are shared among N_a applications through an election process with synchronized conflict resolution over a communication network with diameter D. If the communications occur over a reliable channel and the function (5) maximized during the election routine is submodular, then DRAGON converges in a number of iterations bounded above by $N_a^2 N_v D$.*

Proof: (sketch) We first show by induction that agents agree on the first k assignments in at most kN_aD iterations. Given the submodularity of z_n, the assignment $(i_1^\star, n_1^\star)$ with the highest vote computed at iteration 1 can be outvoted at most $N_a - 1$ times, i.e., until every agents voted on node $n_1^\star$ at least once. Since each time D iterations are needed to propagate the vote, every agent will have agreed on the highest vote $v_{i_1^\star n_1^\star}$ at most after N_aD iterations. Let us suppose that at iteration hN_aD all agents agree on the first k-best assignments. Since the next-best vote propagated at iteration $k+1$ can be outvoted at most $N_a - 1$ times, it follows that every agent will have agreed on $(i_{h+1}^\star, n_{h+1}^\star)$ by iteration $hN_aD + N_aD$. Then, together with $(i_1^\star, n_1^\star)$ being agreed to at N_aD, every agent will have agreed on $(i_k^\star, n_k^\star)$ within kN_aD iterations. In DRAGON each compute node may be assigned to each application, then, in the worst case there is a combination of N_aN_v assignments. Therefore, agents reach agreement in at most $N_a^2 N_v D$ iterations. ∎

As a direct corollary of Theorem VI.2, we compute a bound on the number of messages that applications have to exchange in order to reach an agreement on resource assignments. Because we only need to traverse the communication network at most once for each combination applications per hosting nodes $(i, n) \in \mathcal{I} \times \mathcal{N}$, the following result holds:

Corollary VI.2.1. *(DRAGON Communication Overhead). The number of messages exchanged to reach an agreement on the resource assignment of N_v nodes among N_a non-failing applications with reliable delay-tolerant channels using the DRAGON algorithm is at most $N_{msp} N_a^2 N_v D$, where D is the diameter of the communication network and N_{msp} is the number of links in its minimum spanning tree.*

Performance Guarantees. The election routine in DRAGON is trivially extended with partial enumeration [14], leading to the following two results (for brevity, the extension has been omitted in Algorithm 4).

Theorem VI.3. *(DRAGON Approximation Bound). DRAGON extended with partial enumeration yields an $(1 - e^{-1})$-approximation bound with respect to the optimal assignment.*

Proof: (sketch) During the election routine, DRAGON uses a greedy heuristic to assign node resources to a set of winners $\mathcal{W}_n$. The objective of the heuristic is to maximize the value of the set function $z_n(\mathcal{W}_n)$ without exceeding the node capacity (knapsack constraint). From a recent result on submodular functions [15], we know that a greedy approximation algorithm used to maximize a non decreasing submodular set function subject to a knapsack constraint is bounded by $(1 - e^{-1})$ if the algorithm is combined with the enumeration technique due to [14]. Being the set function $z_n(\mathcal{W}_n)$ positive, monotone and non-decreasing, it remains to show that the overall utility maximized by DRAGON is submodular, which comes from Lemma VI.1; hence the claim holds. ∎

Theorem VI.4. *(DRAGON Approximation Optimality). The DRAGON approximation bound of* $(1-e^{-1})$ *is optimal, unless* $NP \subseteq DTIME(n^{O(\log\log n)})$.

Proof: (sketch) To show that the approximation bound given by DRAGON is optimal, we first show that the applications-resources assignment problem addressed by DRAGON can be reduced from the (NP-hard) *budgeted maximum coverage problem* [14]. Given a collection S of sets with associated costs defined over a domain of weighted elements, and a budget L, find a subset $S' \subseteq S$ such that the total cost of sets in S' does not exceeds L, and the total weight of elements covered by S' is maximized. We reduce the applications-resources assignment problem from the budgeted maximum coverage problem by considering *(i)* S to be the collection of all the possible set of applications, i.e., $S = 2^{\mathcal{I}}$, *(ii)* L to be the total amount of resources available on the hosting node (in this particular case $N_\rho = 1$), and *(iii)* weights and costs to be votes and demanded resources of each application. Since [14] shows that $(1-e^{-1})$ is the best approximation bound for the budgeted maximum coverage problem unless $NP \subseteq DTIME(n^{O(\log\log n)})$, the claim holds. ■

VII. Evaluation

To validate the approach presented in this paper, we implemented a prototype of DRAGON, available at [16]. Our evaluation focuses on two major sets of results; we first assess both DRAGON's asynchronous convergence properties and performance; then we provide evidence of the advantages derived by using DRAGON analyzing two use cases: cache placement for a CDN provider and process migration for mobile gaming.

A. DRAGON properties evaluation

We evaluate convergence and performance properties of DRAGON over a simulated environment with 4 compute nodes, each with a different amount of computing resources (CPU, memory and storage). We run 6 diverse services, whose implementation can be chosen among 9 different functions; on average, each function uses about 13% of a node capacity. These numbers, combined with the rest of our parameter space, allowed us to test the algorithm behavior after the hosting resources are saturated, even running a moderate number of applications. All tests have been repeated varying the number of concurrent applications.

Convergence Evaluation. DRAGON convergence properties have been evaluated by measuring the time needed to reach consensus and the total number of messages exchanged. To stress the convergence of the algorithm, we simultaneously deploy up to 20 applications, i.e., all DRAGON agents begin the execution at the same time. Figure 2ab shows our results comparing three system policies: *(i)* services of an application are preferably allocated on the lowest number of nodes; *(ii)* services of an application are spread across as many nodes as possible; *(iii)* no preference on the number of nodes is given. For each configuration, we ran 25 instances, gradually varying the average number of services per application (with averages from 2.4 to 3.6 services). Plots show mean values; all confidence intervals (not shown) were statistically significant.

In particular, Figure 2a shows the mean convergence times. We found that, when a large number of applications interact, encouraging the system to use fewer nodes significantly lowers convergence time. Some consequences of this policy are *(i)* a reduced probability to lose a node election and *(ii)* re-voting on residual resources located on additional nodes is discouraged. Hence, the highest convergence times have been registered enforcing the usage of many nodes, while convergence is slightly faster when applications are free to arbitrarily decide the number of nodes to use.

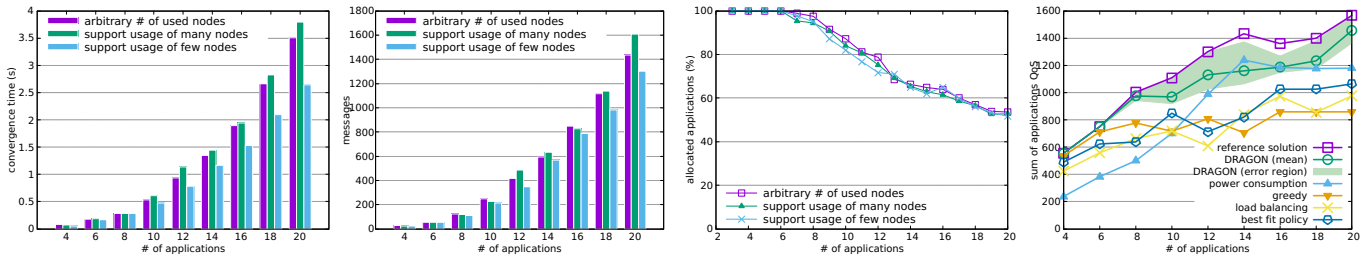
The total number of exchanged messages follows a similar behavior (Figure 2b). However, in this case the previous trend is evident only when the number of applications is greater than 18 and the difference among values of different policies is not marked as for convergence times. Thus, changing this policy does not seem to significantly impact the number of messages that DRAGON needs to exchange to reach convergence.

Performance Evaluation. Figure 2c compares DRAGON performance for the same three system policies previously introduced. The plot shows the percentage of applications successfully deployed after the distributed assignment process. We found that, when the number of concurrent applications stays below 8, all requests are allocated, since the overall resources demand is bounded by the total amount of available resources.

Above that threshold, the percentage of allocated bundles starts to gradually decrease. All analyzed policies achieve approximately the same average allocation ratio, with the only exception of the "few-nodes-policy", which obtains a lower allocation ratio when the number of applications is between 8 and 12, although it shows the fastest convergence time (Figure 2ab). This is because, when resources on the already used nodes terminate, this policy discourages the usage of residual resources available on other nodes. However, this disadvantage disappears as the number of applications grows, since the system implicitly introduces more allocation options. This result suggests that DRAGON allocation ratio scales well with the application concurrency regardless the system policy.

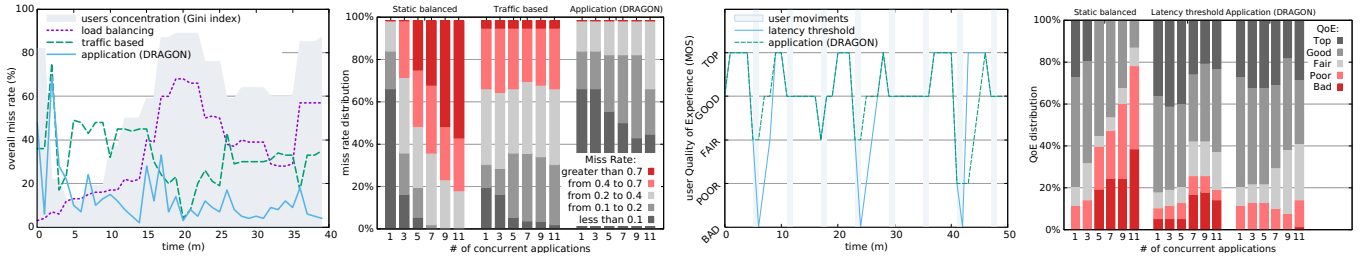
At last, to evaluate our performance in practice we compared DRAGON with traditional orchestration approaches. In particular, we compare against three one-size fits-all allocation policies, i.e., a centralized orchestrator that uses the same objective function to optimize the deployment of all applications: *(i)* minimization of total *power consumption*, *(ii) greedy* selection of the potentially highest performant functions, *(iii) load balancing* among nodes. Figure 2d also shows performance obtained switching between these three policies based on which one *fits best* each application needs. Additionally, we plot the *reference solutions*, obtained running a centralized solver to the Problem 1. Values obtained with this experiment set have been used as reference to evaluate the other approaches.

Figure 2d compares solutions in terms of *overall Quality of Service*, i.e., the sum of the QoS obtained by each application



(a) Convergence time. (b) Total number of exchanged messages. (c) Percentage of allocated bundles. (d) Sum of deployed applications QoS.

Fig. 2: (ab) Convergence evaluation of DRAGON for different system policies. (cd) Performance evaluation of DRAGON comparing (c) different system utilities and (d) DRAGON solutions against *(i)* three one-size fits-all common approaches and *(ii)* a reference solution obtained running a centralized solver.



(a) Miss rate over time. (b) Miss rate distributions. (c) QoE over time. (d) QoE distribution.

Fig. 3: (ab) Evaluation of a CDN cache provisioning application comparing different deployment strategies: (a) miss rate over time varying the geographical users distribution; (b) distribution of measured miss rate varying the number of concurrent applications. (cd) Evaluation of a mobile gaming application for different deployment strategies: (c) QoE over time for an user moving in different areas; (d) QoE distribution varying the number of concurrent applications.

successfully deployed[1]. Varying the number of concurrent applications, for each configuration we ran DRAGON multiple times. Results are shown with a 95% confidence interval. Centralized algorithms have been run once, as they give always the same solution. Results show that allowing each application to deploy its services according with its own objectives through DRAGON provides a considerably higher QoS compared to one-size fits-all approaches, despite being a distributed algorithm. In particular, for less than 8 applications, i.e., before resources start to run out, DRAGON is always reference solution, considered as optimal. For an higher number of distributed instances, as expected, the mean QoS starts to degrade departing from the optimal. However, the total QoS continues to grow, following the trend of the reference solution. This result suggests that DRAGON effectively prefers new applications that introduce higher utilities to the overall solution.

Other findings from Figure 2d are summarized as follows. *(i)* A common objective that minimizes the overall power consumption provides poor total QoS, except for an high number of applications, since this strategy accommodates the largest number of requests. *(ii)* Greedily selecting the most performant physical functions provides high values of overall QoS only when there are few applications. Finally, *(iii)* switching among different common strategies based on the one that fits best each application does not necessarily provide an higher QoS. This is because these allocation strategies work well when they are applied to all applications in the same way (e.g., load balancing and power consumption minimization). Noticeable, none of the one-size fits-all approaches is able to increase the overall QoS after resources are saturated.

[1]The QoS of each application have been modeled through its private utility. This provides us a qualitative parameter to compare solutions. QoS values have been normalized between 0 and 100 for each physical function.

B. *Reference use case evaluation*

We now show the advantages brought by DRAGON when adopted by an infrastructure provider that wants to deploy application requests coming from multiple service providers, without restricting them on the deployment approach. Our aim is to show that, for example, a CDN provider that relies on a third party infrastructure to serve a certain area may benefit from using its own cache placement algorithm (e.g., [17]), running over DRAGON, rather than depending on a one-size fits-all embedding orchestrator.

We setup a simulated environment to evaluate two different edge use cases from [3], [4]: *(i)* cache placement for a CDN provider [4], and *(ii)* edge migration for mobile gaming [3]. In our tests, we compared the provided QoS resulting from different deployment approaches, also varying the concurrency level adding some concurrent applications, thus evaluating the behavior when resources start to become scarse.

CDN Caches. A CDN provider provisions content caches over an edge network where users density dynamically changes across compute nodes. The objective of the provider is to minimize the average miss-rate occurring on deployed caches. The CDN application should react to events where a significant set of users shifts from a node to another. In our tests we simulated a set of 100 users moving over a network of 10 edge compute nodes. To understand how users are distributed among nodes, we also report the Gini index (an high index indicates that most users are located near few host nodes). We summarize our findings in a few take home messages:
(i) A one-size fits-all approach that places caches balancing the resource consumption per node, achieves good performance when users are well distributed, but the miss-rate grows fast

when the concentration increases (Figure 3a). A similar result is obtained by statically partitioning the resources among coexistent applications (Figure 3b) when their number is high with respect to the available resources.
(ii) A one-size fits-all approach that places caches according with the traffic load on each node, achieves optimal miss-rates when users are concentrated on few nodes, while performance is poor otherwise. This is because a low traffic amount on a certain node does not necessarily mean that users are consuming less variety of contents. Figure 3b shows a slight degradation increasing the number of concurrent applications.
(iii) If the application can place caches based on current miss-rate on each nodes, mandating resource partitioning to DRAGON, optimal miss-rate both for low and high users concentration is achieved (Figure 3a). Moreover, note how Figure 3b does not show a noticeable QoS degradation increasing the number of concurrent applications, showing the scalability of our approach. This is because DRAGON seeks optimal resource partitioning with regard to the application objectives.
Mobile Gaming. A gamer moves into an area served by multiple edge nodes. Whereas the application may consider relocating (part of) the game edge functions to better fulfill the latency requirements, the relocation may happen in a crucial phase of the game, causing undesirable service degradation [3]. Therefore, if the deployment is managed by the gaming application itself, it may recognize the time frame in which a relocation is most appropriate (e.g., after the gamer reaches a checkpoint or during the loading of a new level).

In our tests we simulated an user moving every 6 minutes across a network of 10 edge nodes. We measured the Quality of Experience perceived by the user based on latency and packet loss, using the same Mean Opinion Score (MOS) described in [18] for medium-paced games. Our findings are summarized as follows (Figure 3cd):
(i) Statically partitioning resources between applications does not scale (Figure 3d): the application may be unable to migrate services on needed nodes, since resources are assigned to other peers, despite not being currently used.
(ii) If the resources are managed by a one-size fits-all orchestrator that minimizes the end-to-end latency, the user may often experience a QoE level that we label as `bad`, due to some process relocation occurring during the game session (Figure 3c). Figure 3d shows that the percentage of `bad` QoE measurements even may increase with the concurrency.
(iii) If the relocation decision is taken by the application, and resources are dynamically assigned with DRAGON, the service is not migrated rapidly whenever the user moves away; even if this may temporarily increase the latency, it prevents undesirable service degradation during a game session and the overall perceived QoE results improved (Figure 3c). Figure 3d also shows that this approach scales well with the number of concurrent applications.

VIII. Conclusion

This paper proposes DRAGON, a distributed bounded approximation algorithm that solves the problem of optimally partitioning a pool of resources between multiple edge applications. DRAGON allows such applications to coexist over a shared infrastructure by means of a dynamic agreement on which resources have to be (temporary) assigned to which application. We used linear programming to define and model the application-resources assignment problem, that DRAGON solves in a distributed fashion providing guarantees on both convergence time and performance. Our evaluation assesses convergence and performance properties, comparing different policies of our system. Moreover, we evaluate DRAGON over two representative edge use cases, showing that an infrastructure provider may adopt it to enable their customers (i.e., service providers) to deploy applications through the preferred embedding algorithm, without the restrictions deriving by relying on a common one-size fits-all orchestrator.

Acknowledgment

The work of Gabriele Castellano was conducted as visiting scholar in the Computer Science Department at Saint Louis University. The authors would like to thank Tierra Telematics and NSF CNS-1647084 for their support.

References

[1] ETSI, "NFV MANO." [Online]. Available: https://goo.gl/XLffVJ
[2] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.
[3] V. Sciancalepore *et al.*, "A double-tier MEC-NFV architecture: Design and optimisation," in *Standards for Communications and Networking (CSCN), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–6.
[4] T. Taleb *et al.*, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, pp. 1657–1681, 2017.
[5] E. Amaldi *et al.*, "On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 213–220, 2016.
[6] S. Mehar *et al.*, "An optimized roadside units (rsu) placement for delay-sensitive applications in vehicular networks," in *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*. IEEE, 2015, pp. 121–127.
[7] T. Bahreini and D. Grosu, "Efficient placement of multi-component applications in edge computing systems," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 5.
[8] N.-S. Vo *et al.*, "Optimal video streaming in dense 5g networks with d2d communications," *IEEE Access*, vol. 6, pp. 209–223, 2018.
[9] F. Esposito, "Catena: A distributed architecture for robust service function chain instantiation with guarantees," in *Network Softwarization (NetSoft), 2017 IEEE Conference on*. IEEE, 2017, pp. 1–9.
[10] B. Hindman *et al.*, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, vol. 11, no. 2011, 2011, pp. 22–22.
[11] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.
[12] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm." in *USENIX Annual Technical Conference*, 2014, pp. 305–319.
[13] N. A. Lynch, *Distributed algorithms*. Elsevier, 1996.
[14] S. Khuller *et al.*, "The budgeted maximum coverage problem," *Information Processing Letters*, vol. 70, no. 1, pp. 39–45, 1999.
[15] M. Sviridenko, "A note on maximizing a submodular set function subject to a knapsack constraint," *Operations Research Letters*, vol. 32, no. 1, pp. 41–43, 2004.
[16] G. Castellano, "DRAGON Prototype," https://github.com/gabrielecastellano/dragon, 2018.
[17] D. Karger *et al.*, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. ACM, 1997, pp. 654–663.
[18] M. Jarschel *et al.*, "An evaluation of QoE in cloud gaming based on subjective tests," in *Fifth conference on Innovative mobile and internet services in ubiquitous computing (imis)*. IEEE, 2011, pp. 330–335.